Natural Spontaneous Order in Wireless Sensor Networks: Time Synchronization Based On Entrainment

Aggelos Bletsas and Andrew Lippman
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: {aggelos, lip}@media.mit.edu

***Abstract*— Time is critical for a variety of applications in wireless sensor networks. In this work we present, theoretically analyze and evaluate in practice a simple time synchronization algorithm for wireless sensor networks, based on entrainment. This algorithm requires no specialized servers or beacons but it relies on local communication between neighboring nodes spontaneously emerging to global network synchrony.**

Its simplicity, accuracy and precision are examined through the practical implementation of tho different kinds of wireless sensor networks. Error synchronization error on the order of microseconds is observed 100% of the time, with minimal communication and computation cost tailored to the available resources of the wireless sensor network nodes.

I. Introduction

A common time reference is an essential piece of information for a variety of pervasive computing and communication applications. In the special case of Wireless Sensor Networks, common time keeping among the members of the network is not only essential but also critical since information distributively and collectively gathered, needs to be time stamped so it can be correlated and processed. For example *information beam-forming* facilitated in sensor networks relies on a common time reference. Moreover *security* schemes in wireless networking require time-synchronization (see for example the Tesla authentication algorithm [8]). Finally, the whole argument behind the *energy savings* of wireless multi-hop communication between any two points, compared to the single-hop case, is based on the assumption that there is coordination among the relay nodes between transmitter and receiver i.e the relay nodes are listening during the transmission, otherwise packets are lost and information needs to be resent. A common clock could assist that necessary coordination by assuring that the intermediate relay nodes "wake up" or "go to sleep" at the correct time intervals in ways that information is relayed and energy savings are realized [2].

Apart from the aforementioned cases, a common time reference is also important in range estimation using time-of-flight measurements of acoustic or radio frequency signals and consecutively in triangulation and *location determination*. Location awareness is considered an important aspect of future wireless sensor networks [4].

Even though the significance of time keeping in sensor networks is prominent, researchers in the field have traditionally bypassed the problem. The emergence of low-cost Global Positioning System (GPS) receivers is believed to provide an adequate solution. However that is clearly a misconception: GPS has coverage inefficiencies at indoor, underwater or extraterrestrial environments and there is significant time left until GPS receivers cost less than the micro-controller unit of the wireless sensor node, especially when inexpensive and low capability micro-controllers are used. Moreover a satellite timing and positioning service like GPS is conceptually unnecessary since the wireless sensor network deployed could provide itself for the critical timing services.

In the following sections we will describe how such services could be autonomously facilitated. In this work we have focused on a completely distributed, server-free and decentralized approach: all the nodes in the network are homogeneous, they are running the same time synchronization algorithm which is a part of their overall sensing and communicating task i.e the nodes are not exhausting their computational and communication resources for time synchronization. We were inspired by similar mechanisms found in nature: the way fireflies manage to globally blink in unison, even though they interact locally or the way millions of cardiac neurons fire in sync to control our breathing. We were particularly attracted by their global effect of sync that emerged as a consequence of local interactions between homogeneous elements (fireflies or cardiac neurons). Those are canonical examples of *Entrainment*. For a charming description of relevant research on spontaneous order in natural phenomena, the interested reader should refer to [9]. In the above examples of entrainment, synchrony is not controlled by any centralized authority but it is the natural emergent result of local interactions. This is in contrast with centralized solutions to synchronization which are based on central servers [1], [7] or specialized beacons [3].

A. Desiderata for Time Sync in Wireless Sensor Networks

Before proceeding to the description of the technique, we should emphasize the criteria upon which every time sync

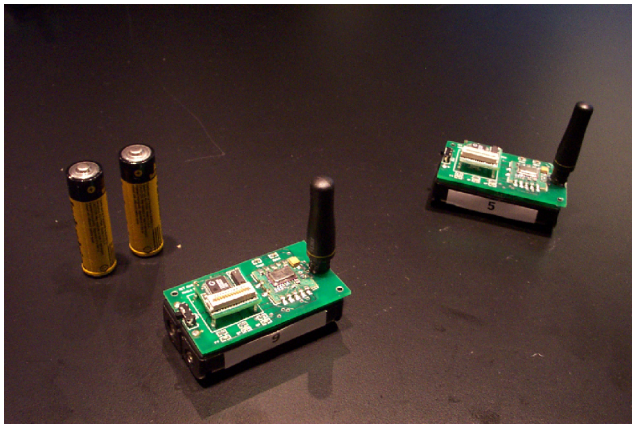


Fig. 1. rfBeatles: the wireless sensor network nodes created.

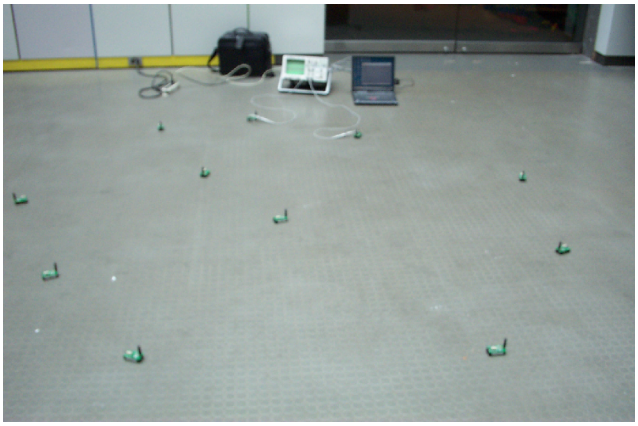


Fig. 2. The experimentation setup with the rfBeatles deployed.

algorithm for wireless sensor networks should be evaluated.

- accuracy/precision: the time sync error between any two nodes of the wireless sensor network (accuracy) and how often this error is realized (precision). In this work, we aimed for synchronization error on the order of μsecs (10 μsecs error in time results in approximately 3.4 millimeters error in range estimation when acoustic signals are used).
- communication cost: given the fact that communication is energy expensive, the bandwidth used for the exchange of timing information among the network nodes, for any desired accuracy and precision, should be quantified.
- computation cost: sensor nodes are usually equipped with relatively "lightweight" hardware. Any necessary computation should be tailored to the available resources (limited memory and computation speed). For example, the implementation of a kalman filering technique in a 8-bit micro-controller would be insufficient.
- complexity/scalability: it is important to understand that even the simplest algorithm needs to be implemented in a network of nodes rather than a pair of nodes. Therefore scaling and complexity issues need to be resolved especially when the number of networked nodes increases. In a centralized server or beacon-based approach any exchange of information between two nodes might require sophisticated communication and network routing protocols that exhaust the available energy, bandwidth and computation resources at each node and therefore such approaches might be inappropriate for wireless sensor networks.

In section II we describe our decentralized, entrainment-based approach while in section III we present the evaluation results according to the above criteria, over two kinds of sensor networks that we implemented. We conclude in section IV.

II. Technique Description

Our goal was to provide a synchronization algorithm that is based in local communication between neighboring nodes and time synchrony emerges as a global network attribute even between wireless nodes that are not in range (but are connected through the network). That "global-from-local" attractive property of *entrainment* found in complex natural systems, like those described in the previous section could meet the complexity/scalability specification.

Interestingly, Lamport in his seminal work [5] described a synchronization algorithm for computer clocks/processes in the context of computer operating systems with that important scalable "global-from-local" property. His algorithm was based on the fact that *Time* is a strictly monotonically increasing quantity, therefore events happening in subsequent times should have timestamps ordered accordingly, otherwise a correction in the clocks should be made.

In this work, a) we customize Lamport's algorithm to Wireless Sensor Networks, bearing in mind that individual nodes are usually equipped with primitive hardware (for example 8-bit micro-controller as the basic CPU and no other memory) and b) we quantify the synchronization error both in theory and practice through two different implementations of real world sensor networks. To our knowledge, the solution provided for time synchronization in wireless sensor networks and its verification through concrete implementations are so far unique. The entrainment-based algorithm for each node in the sensor network is provided below:

- **broadcast:** broadcast your timing information every δT seconds. The timing information is your own clock time $C(t)$ upon read plus the transmission time $1/R$ of the packet that includes the timing information (R in packet/sec is the speed of the communication link): $C_b(t) = C(t) + 1/R$

- **receive and compare:** upon reception of timing information $C_b(t)$ from a neighboring node, read your own clock $C(t)$ and compare them. If $C(t) < C_b(t)$, then replace your clock value with the received information $C_b(t)$: $C(t) \leftarrow C_b(t)$.

It is important to note that the above algorithm can be implemented in a completely peer-to-peer fashion, without the

need of a specialized protocol. Timing information could be *piggybacked* in every message exchanged between neighboring nodes converging to global absolute time synchronization. This is a great simplification over the cases where a specialized server is used to disseminate timing information and therefore network routing should also be implemented. Even when compared to the cases where specialized beacons are used, the above algorithm is advantageous since it ensures synchrony provided that there is communication between one node and at least one of the rest of the network nodes, while the beacon approach requires in principle all the nodes to be in communication range with the broadcasting beacon, otherwise nontrivial coordination between multiple beacons is necessary increasing the complexity of the system.

The peer-to-peer nature of the algorithm, its "global-from-local" character, its convergence speed (analyzed later) and its resemblance with synchrony examples found in nature (as described at the introduction), nominated the above algorithm as *natural, spontaneous time synchronization, based on entrainment*.

Before proceeding to the analysis of this algorithm, we ought to emphasize practical considerations on its application to wireless sensor networks.

A. Practical Considerations

Since we are aiming at a synchronization error on the order of μseconds, it is imperative to have clock resolution less than 1 μsecond. That is feasible when the wireless sensor network node incorporates an oscillator with higher than 1 MHz frequency. The clock value $C(t)$ could be represented by a software variable which is incremented when a counter overflows. The resolution of the counter should be below 1 μsecond. The length of the time variable and the length of the counter (8-bit, 16-bit, 32-bit, 64-bit etc) depends on the phenomenon the sensor network monitors. For example, for a 22.1184 MHz oscillator and a 16-bit counter incremented by the system oscillator tick divided by 12, the time resolution of the pair time variable/counter is $\frac{12}{22.1184} = 0.54$ μseconds and an eight-bit time variable overflows after approximately 9 seconds.

Under the above practical implementation details the pair time variable/counter that represents $C(t)$ is no longer a strictly monotonically increasing variable but it zeroed after a specified period of time. Fortunately, that can be addressed with a slight modification of the algorithm at the **receive and compare** phase. The local clock value $C(t)$ is compared to the received value $C_b(t)$ only if $C(t)$ is sufficiently high so it has "escaped" from the overflow state. Practically that means that $C(t)$ is always compared to a threshold value and if it is bigger, then the **receive and compare** stage of the algorithm is executed. In that way, unnecessary timing oscillations are avoided that would last until all the clock variables across the network had overflowed.

Another interesting observation is that the comparison in the second stage of the algorithm, also needs some time. Moreover, the overall reception and processing of a packet also needs time. That amount of time should be taken into account, especially when high level programming languages are used (for example C instead of Assembly). Practically, that means that $C_b(t)$ should be increased appropriately by a factor determined through measurements. Operating system intricacies should be less of a problem given the fact that wireless sensor nodes operate with customized software and hardware.

B. Theoretical Analysis

The following theorem quantifies the maximum synchronization error of the proposed algorithm. It is important to add that synchrony is achieved (and therefore the error below is realized) within a single packet exchange, between two nodes in communication range. When the two nodes are d hops away then in principle d packets need to be exchanged. A similar result is reported in [5].

Theorem 2.1: The maximum time synchronization error e between two wireless sensor network nodes equipped with clocks having frequency skew (frequency offset) ϕ, employing radios with range R meters, communicating in range R every δT seconds with S packets/sec transmission speed and employing the Spontaneous Time Sync algorithm is given by

$$\epsilon = \phi\,\delta T + R/c \tag{1}$$

where c is the propagation speed of the communication signal (i.e. $3\,10^8$ m/sec for RF, 340 m/sec for sound etc.) For a wireless network of the above nodes with maximum number of hops (diameter) d, the maximum error e is given by

$$\epsilon = d\,(\phi\,\delta T + R/c) \tag{2}$$

Observe that the error is independent of the transmission speed S of the communication link since that parameter is taken into account by the Spontaneous Time Sync algorithm.

Proof: A simple and intuitive proof is provided at the Appendix. A similar result with a more difficult proof could be found in [5].

It is important to note that the above error equations reveal precisely the balance between time synchronization error (ϵ), stability of wireless sensor node clocks (ϕ) and bandwidth spent for timing messages ($1/\delta T$). The more stable clocks used (smaller frequency skew ϕ), the more often timing messages are exchanged (smaller δT), the smaller the error (ϵ) becomes.

III. Experimental Evaluation

Two kinds of wireless sensor networks were built to test the Spontaneous Time Sync algorithm, one based on radio frequency (RF) communication and one based on infrared. In the first case temperature was sensed (even though other qualities could be measured) and in the second case the goal

was distributed playback of music (audio information) and synchronization of two displays at the edges of the network (visual information).

A. RF Network

In that setup, each node is equipped with a mixed signal 32 KByte ROM/2 KByte RAM 8-bit micro-controller connected to a 22.1184 MHz crystal oscillator. The micro-controller pins are routed towards two connectors, one below the board where the communication module is connected and one on the top side of the board where various sensing circuits (application modules) could be connected. That micro-controller, oscillator and two connectors consist of the "Pushpin" processing layer, described in detail in [11] and introduced in [6]. We chose the Pushpin processing layer, since its 2-connector stacked architecture allows flexible integration of various custom communication modules and application sensing circuits at a relatively small cost.

For the RF implementation (figure 1) we designed a RF communication layer based on the 916.5 MHz radio TR1000 of RF Monolithics. Directly connecting the Received Signal Strength (RSSI) pin, transmit pin (Tx) pin and receive pin (Rx) to the ADC, DAC and one interrupt-driven input pin respectively of the Pushpin micro-controller, we easily created a lightweight, embedded software-defined radio, powerful enough for wireless sensor networking, at a small cost (less than 40$ in small quantities). Figure 1 displays the wireless sensor network node created that we call "rfBeatle".

Custom software modules that we developed provided 50 kbps point-to-point communication while there was transmission range control through the output voltage. Rudimentary error control functions were implemented (for example there was cyclic redundancy check on every packet) and for short ranges the throughput could be doubled since higher SNR could provide for 2-bit Pulse Amplitude Modulation (PAM). Apart from the communication modules, the software at each node implemented the Spontaneous Time Synchronization algorithm using a 16-bit counter which incremented at overflow an 8-bit time variable.

Therefore the timing information totally consisted of 3 bytes, representing the clock $C(t)$ for each node. The total timing packet that was broadcasted, consisted of 4 bytes with the last one containing the CRC of the previous three bytes of $C(t)$. During the implementation of the algorithm all the practical consideration presented in II-A were taken into account (for example the second stage of the algorithm was executed only when $C(t)$ was sufficiently high, away form the reset value). Observe the total lack of any other excessive information, for example there is no routing information: for the purpose of time synchronization, each node broadcasts only its timing information (as defined in the broadcast phase of the algorithm) with an additional error check code (CRC). Since we didn't want to exhaust the computational resources for the timing algorithm, each node was set to broadcast temperature information every 5 milliseconds. That rate might seem excessive, however we wanted to show that the time sync

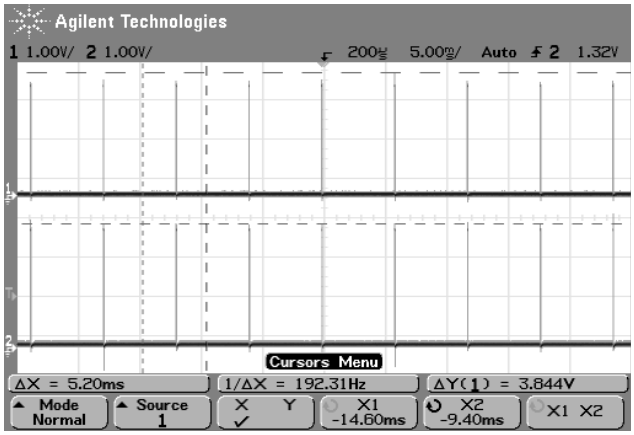


Fig. 3. 2-channel oscilloscope trace (one channel per node.)

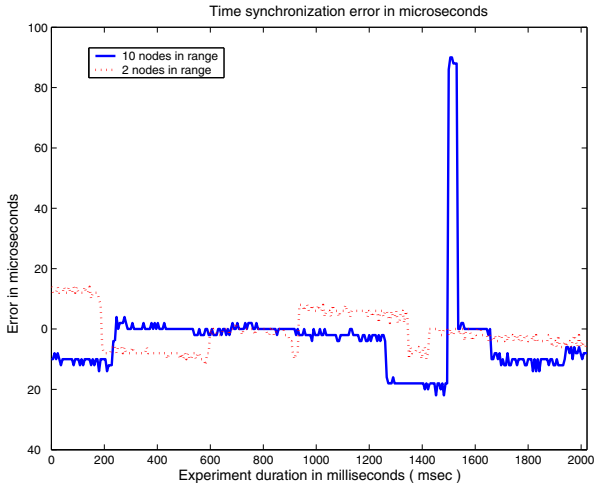
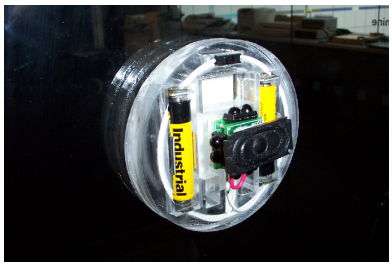


Fig. 4. Time Synchronization in μsecs.

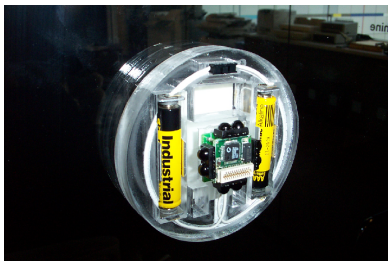
algorithm was insensitive to increased computational loads at each node.

In figure 2 the experimental setup is displayed. Ten rfBeatles are running the Spontaneous Time Sync algorithm as described above and two of them are connected to a digital oscilloscope. The nodes are outputting a pulse at a specified pin at specified time instants (every 5.2 msecs). The phase difference of those pulses reveals the time synchronization error. The digital oscilloscope is connected to a laptop where the 2-channel trace (figure 3) can be downloaded and analyzed.

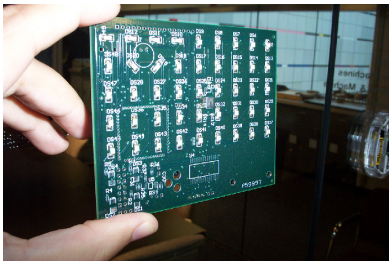
1) Quantitative Results: Figure 4 depicts the synchronization error over a span of 2 seconds, for a pair of communicating nodes and a network of 10 nodes. δT is approximately 100 msecs and (nominal) maximum frequency skew ϕ of the crystals is 100 parts-per-million (ppm). It can be seen from the plot that the error is less than 20 μseconds, most of the time (we will quantify the precision later). In the case of 2 nodes in range, the error is on the order of 10 μseconds or less while in the case of 10 nodes, there are periods where the error is



(a) 4-IR Pushpin with speaker.

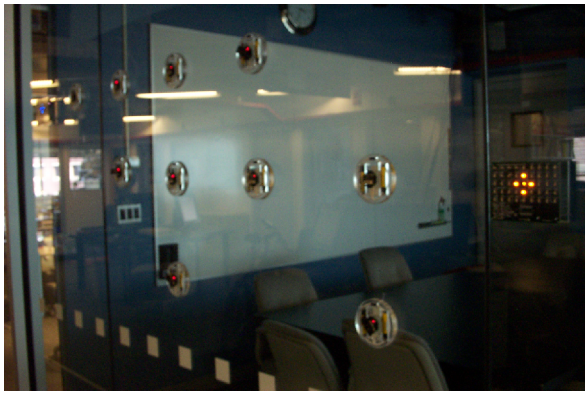


(b) 4-IR Pushpin without speaker.



(c) 45-LED display.

Fig. 6. Audio and Visual output for the infrared-based network



(a) Left side view of the infrared network.

(b) Right side view of the infrared network.

Fig. 7. Wall installation of the infrared-based network.

on the order of 4 μseconds since the larger number of nodes results in a higher rate of timing information broadcasting (larger than $1/\delta T$). There is also a "glitch" of approximately 90 μsecs which could be attributed either to the rudimentary CRC function or to a less stable oscillator in the set of 10 nodes.

The error cumulative distribution function in figure 5 clearly shows that the synchronization error is smaller than 12 $\mu secs$ for 98% of the time for a pair of nodes and less that 25 $\mu secs$ for 98% of the time for the case of 10 nodes. It is important to note that for the above values of $\phi, \delta T$ and $R \leq 100$m, the error e should be on the order of 10 μsecs. From figures 4, 5 we can see that this theoretical bound is slightly exceeded, fact that can be justified by computational delays at each node which are not included in the theoretical model (see relevant discussion in section II-A). Nevertheless, the smaller than 25 μsecs error (98% precision) *for the hardware and bandwidth used*, is impressive.

B. Infrared Network

In this case, the RF communication layer is replaced by the original infrared communication module of Pushpins [11]. Custom 32 kbps communication routines were developed and the whole node was packaged in a plexi-glass puck (figure 6(b)). The nodes were placed in a canonical grid on a wall, bearing in mind the directivity of the 4-way infrared links (figure 7).

The goal was to implement the Spontaneous Time Synchronization algorithm in a case where not all the nodes are in direct communication (diameter of the network $d > 1$). That was easier with infrared communication due to its short range and directivity and more difficult with RF. The output would be distributed music playback: the speaker-equipped nodes would playback the same piece of music that would be listened in sync if and only if the wireless nodes were in sync. To emphasize the network effect of the algorithm i.e. the fact that nodes become synchronized even though they not in direct communication but they are only connected through a multi-hop network, we connected two infrared-equipped displays at the edges of the network. The displays should visualize the same (changing) motif in synchrony, even though they are not directly in communication range but they are communicating through the infrared network. The displays consisted of 5x9 LEDs that were connected to a LED-display driver wired to an infrared-equipped pushpin through its serial peripheral interface (SPI) port. Those low cost displays were designed

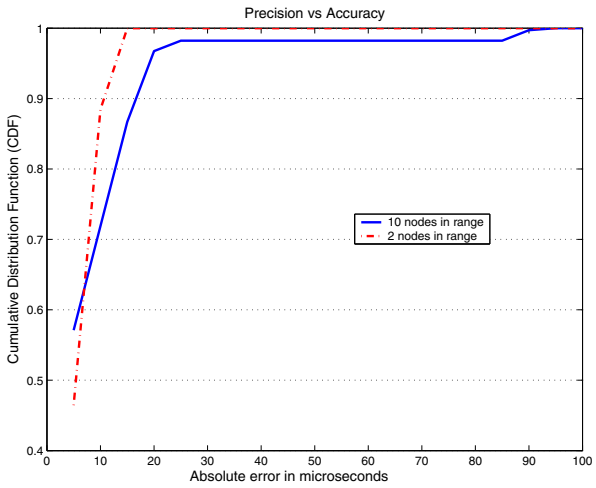


Fig. 5. Precision vs Accuracy.

for the Badge project by M. Laibowitz [10].

The Spontaneous Synchronization algorithm managed to synchronize the speaker and display equipped Pushpins giving the impression to the listener/viewer that the pucks were hardwired. Qualitative measurement were not made, however the error should be less than 1 msec, given the capabilities of the human audiovisual perception.

IV. Conclusion

We presented, theoretically analyzed and practically verified a simple time synchronization algorithm for wireless sensor networks, based on entrainment. This algorithm requires no specialized servers or beacons but it relies on local communication between neighboring nodes spontaneously emerging to global network synchrony.

Acknowledgment

The authors would like to thank Josh Lifton and Mat Laibowitz from the Responsive Environments Group (http://www.media.mit.edu/resenv/) for their kind hardware assistance throughout this work.

References

[1] A. Bletsas, *"Evaluation of Kalman Filtering for Network Time Keeping"*, Proceedings of IEEE International Conference on Pervasive Computing and Communications (PERCOM), Fort-Worth Texas, March 2003.

[2] A. Bletsas, A. Lippman, *"Efficient Collaborative (Viral) Communication in OFDM Based WLANs"*, ITS International Symposium on Advanced Radio Technologies (ISART), Institute of Standards and Technology, Boulder Colorado, March 2003.

[3] J. Elson, L. Girod, D. Estrin, *"Fine-Grained Network Time Synchronization using Reference Broadcasts"*, Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI), Boston MA, 2002.

[4] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, *"Next Century Challenges: Scalable Coordination in Sensor Networks"*, Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MOBICOM), Seattle Washington, August 1999.

[5] L. Lamport, *"Time, Clocks, and the Ordering of Events in a Distributed System"*, Communications of the ACM, vol. 21, no.7, July 1978.

[6] J. Lifton, D. Seetharam, M. Broxton, J. Paradiso, *"Pushpin Computing System Overview: a Platform for Distributed, Embedded, Ubiquitous Sensor Networks"*, Pervasive 2002, Proceedings of the Pervasive Computing Conference, Zurich Switzerland, August 2002.

[7] D. L. Mills, *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, RFC 1305, University of Delaware, March 1992.

[8] A. Perrig, R. Canetti, D. Tygar, D. Song, *The TESLA Broadcast Authentication Protocol*, RSA CryptoBytes, 5(2):2-13, 2002.

[9] S. Strogatz, *Sync, The Emerging Science of Spontaneous Order*, 1st ed. New York: Theia, 2003.

[10] Badge website, http://www.media.mit.edu/resenv/badge

[11] Pushpin website, http://web.media.mit.edu/ lifton/PushPin/

Appendix

Theorem 2.1 proof: we proceed in two phases. Initially, let's assume that a pair of nodes are time synchronized at time t_0 and they are running the Spontaneous Time Sync algorithm. After δT the time synchronization error ϵ between two nodes with clock frequency offset (skew) ϕ becomes $\phi\delta T$:

$$\epsilon\,(t_0 + \delta T) = \phi\,\delta T. \tag{3}$$

The *receive and compare* phase of the algorithm will not happen instantly, but after R/c seconds which corresponds to the propagation delay of the communication signal (transmission delay is already incorporated into the time stamp broadcasted). Therefore immediately after correction ($C(t) \leftarrow C_b(t)$) the time error is R/c. As a result the algorithm's time sync error between the who nodes is $max\{\phi\,\delta T, R/c\}$.

Now, let's relax the initial synchronization assumption. Immediately after the first execution of the *receive and compare* phase of the algorithm, the error synchronization becomes R/c. Just before the next consecutive execution of the *receive and compare* phase, the error becomes $\phi\,\delta T$ larger:

$$\epsilon = R/c + \phi\,\delta T. \tag{4}$$

Now, imagine three nodes A, B, C where B is in between A and C (A and C cannot communicate directly). In this example, the diameter d of the network is 2 ($d = 2$). The maximum synchronization error between A and B is ϵ (equation 4) and similarly, the synchronization error between B and C is again e. Therefore, the maximum synchronization error is $2\,\epsilon$. Generalizing for a series of $d+1$ nodes, it is easy to see that the error becomes:

$$\epsilon = d\,(R/c + \phi\,\delta T). \tag{5}$$

Equation 5 concludes the proof.